VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Testování embedded databázových systémů pro mobilní zařízení

Testing of Embedded Database Systems for Mobile Devices

2018 Petr Šmíd

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Šmíd**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Testování embedded databázových systémů pro mobilní zařízení
Testing of Embedded Database Systems for Mobile Devices

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je testovat embedded databázové systémy pro mobilní zařízení jak z pohledu typu databázového systému (SQL nebo klíč-hodnota), tak z pohledu výkonu a dalších vlastností.

1. Nastudujte problematiku embedded databázových systémů pro mobilní zařízení a proveďte jejich klasifikaci.
2. Vybrané databázové systémy otestujte pro vybrané datové kolekce a vytížení.
3. Proveďte zhodnocení výsledků experimentů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Michal Krátký, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018

.....

Souhlasím se zveřejněním této diplomové práce dle požadavku čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v diplomových programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

.....

Chtěl bych zde poděkovat doc. Ing. Michalovi Krátkému, Ph.D. za vedení této bakalářské práce, odborné rady, konzultace a znalosti, které mi během vypracování poskytl.

Abstrakt

Cílem práce je otestovat vybrané embedded databázové systémy, které jsou určeny pro mobilní zařízení. V rámci práce se čtenář dozví základní informace o jednotlivých typech databázových systémů a budou mu vysvětleny některé základní pojmy z oblasti databází. Čtenáři jsou představeny vybrané testované databázové systémy (SQLite, LiteDB, RealmDB a CouchBaseLite), jejich základní charakteristiky, vzájemné porovnání a je popsán samotný benchmark, kterým jsou vybrané databázové systémy testovány. Hlavním výsledkem práce je srovnání a vyhodnocení výsledků získaných hodnot z testování napříč vybranými systémy.

Klíčová slova: benchmark, embedded, databázový systém, mobilní zařízení, relace

Abstract

The purpose of this thesis is to test four embedded database systems designed for mobile devices. In the first part of the thesis a brief summary is given of the types of database systems and of basic database terminology. The subsequent chapter introduces the database systems that have been chosen for testing (SQLite, LiteDB, RealmDB and CouchBaseLite), their basic characteristics and mutual comparison. Next the thesis continues to describe the particular benchmark that was used to test the above mentioned database systems. The same chapter also describes the logic and principles of the testing methodology. The final chapter of the thesis compares and evaluates the final values gathered during the testing of each of the database systems.

Key Words: benchmark, embedded, database system, mobile device, relation

Obsah

Seznam použitých zkratek a symbolů

XML	–	eXtensible Markup Language
JSON	–	JavaScript Object Notation
SQL	–	Structured Query Language
ACID	–	atomicity, consistency, isolation, durability
XPath	–	XML Path Language
XQuery	–	XML Query
SŘBD	–	Systém řízení báze dat
DBMS	–	Database management system
LINQ	–	Language Integrated Query
ISO	–	International Organization for Standardization
ANSI	–	American National Standards Institute
SEQUEL	–	Structured English Query Language
IBM	–	International Business Machines
DDL	–	Data definition language
DML	–	Data manipulation language
DCL	–	Digital Command Language
TCC	–	Transaction Control Commands
MS-DOS	–	Microsoft Disk Operating System
TPC	–	Transaction Processing Performance Council
OLTP	–	Online Transaction Processing

Seznam obrázků

Seznam tabulek

1 Úvod

Hlavním rozdílem u embedded databázových systémů je, že databáze je v paměťovém prostoru aplikace a odpadá síťová nebo mezi procesová komunikace [1,11,12]. Tyto systémy jsou velmi často využívány tam, kde je třeba mít informace uloženy lokálně a není je zaotřebí okamžitě odesílat na server. Jedná se například o průmyslové měřící nástroje, kdy stačí data sumarizovat z více zařízení v určitém intervalu či mobilní zařízení, u kterých například není zajištěno vždy připojení k internetu. A právě o testování těchto embedded databázových systému určených pro mobilní zařízení je tato bakalářská práce.

První kapitola této práce je věnována teoretické části databázových systémů, ve které je uvedena základní terminologie, která je potřebná pro pochopení této práce. V této kapitole se čtenář dozví, co je to databáze a jaký je rozdíl mezi databází a databázovým systémem. Dále jsou zde vysvětleny vybrané datové modely databázových systémů, které jsou v rámci této práce testovány. V rámci této kapitoly jsou představeny embedded databázové systémy, důvody jejich vzniku a praktické využití. Zároveň je také vysvětlen rozdíl mezi embedded databázovým systémem a databázovým systémem založeným na architektuře klient-server, který je nejvíce využívaným systémem. V závěru první kapitoly je obecně vysvětlen pojem benchmarking a metody testování databázových systémů. V druhé kapitole je vysvětleno, jakým způsobem byly vybrány embedded databázové systémy určené pro mobilní zařízení, které jsou v rámci této práce testovány. Dále následuje představení a srovnáním vybraných systémů. V třetí kapitole je popsán benchmark a samotná aplikace, kterou jsou vybrané embedded databázové systémy testovány. Součástí je také představení prezentační a datové vrstvy. Čtvrtá kapitola je věnována vyhodnocení výsledků z testování. Nejprve je srovnání jednotlivých databázových systémů mezi sebou a dále následuje závěrečné zhodnocení každého databázového systému samostatně. V závěru práce je zhodnocení celého práce a samotnéo testování, včetně přínosu této práce.

2 Základní pojmy

2.1 Databáze

Pojem databáze neboli báze dat, je prostor, kde jsou data organizována a uložena do struktur [1, 8]. Pojem databáze se často zaměňuje s databázovým systém, to ale není správně, neboť databáze je jen jednou ze součástí databázového systému. Pod pojmem databáze si můžeme představit například kartotéku u lékaře, kde jsou karty pacientů organizovány do zásuvek, které jsou označeny písmeny abecedy, jenž značí první písmeno příjmení pacienta. Pokud bychom se bavili o práci zdravotní sestry, která vezme kartu pacienta, doplní do ní určité údaje a poté ji opět založí zpátky do svého umístění, nejedná se již o databázi ale o systém řízení báze dat (dále jen SŘBD).

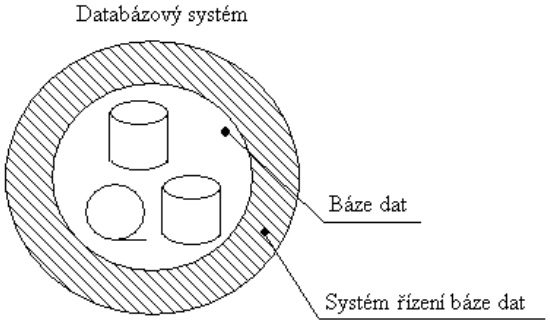
2.2 Systém řízení báze dat

Systém řízení báze dat je program, který je určen pro manipulaci s daty. Tento název pochází z anglického termínu DBMS(DataBase Management System). Mezi základní operace každého SŘBD patří operace pro čtení, vkládání, mazání a aktualizace záznamů. O vykonávání těchto operací se starají tzv. dotazovací jazyky. Nejznámějším dotazovacím jazykem je SQL. Mezi další dotazovací jazyky patří např. XQuery, XPath, které pracují s daty uloženými ve formátu XML. V případě dokumentových databázových systému se část používá LINQ, jenž je součástí frameworku .NET. Další neméně důležitou vlastností SŘBD je definovat strukturu perzistentních dat. Právě tato vlastnost odlišuje SŘBD od jednoduchého souborového systému [1, 8]. Mezi nejznámější SŘBD patří Oracle[1], MS SQL Server[2], Sybase[3], MySQL[4], mSQL[5],PostgreSQL[6], SQLite[7], MongoDB[8], Realm[9], CouchBase[10]...

[1] https://www.oracle.com
[2] https://www.microsoft.com/cs-cz/sql-server/
[3] https://www.sap.com/index.html
[4] https://www.mysql.com/
[5] http://www.hughes.com.au/products/msql/
[6] https://www.postgresql.org/
[7] https://www.sqlite.org
[8] https://www.mongodb.com/
[9] https://realm.io/
[10] https://www.couchbase.com/

2.3 Databázový systém

Databázový systém, jak je vidět na obrázku 1, je spojením databáze a systému řízení báze dat.

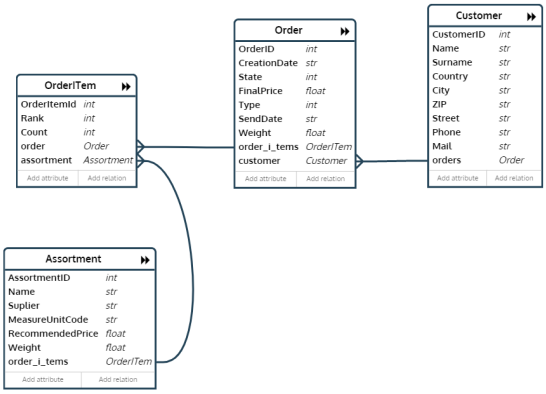


Obrázek 1: Databázový systém [1]

Z obrázku 1 je patrné, že databázový systém je spojením uložených dat a programu, který s těmito daty manipuluje [1]. K tomu aby byl databázový systém kompletní, je třeba do něj zahrnout také uživatelé, kteří s daty manipulují, ať už se jedná o administrátora, který se stará o celý databázový systém, nebo o uživatele aplikace, která s daty pracuje. Většina databázových systémů pracuje na principu klient-server, kdy klientem je aplikace, pomocí které uživatel komunikuje s databázovým systémem, a data která požaduje, jsou získávaná ze serveru. Existují také tzv. embedded databázové systémy, které jsou předmětem této bakalářské práce. Databázové systémy můžeme dále rozlišovat dle toho, zda umožňují více uživatelský přístup či nikoli. Neméně důležitou vlastností je podpora transakcí. Databázové systémy se klasifikují dle datového modelu. Z hlediska způsobu ukládání dat a vazeb mezi záznamy, dělíme datové modely do několika základních typů.

2.4 Relační databáze

Jedná se o nejvíce rozšířené databáze, postavené na relačním modelu dat [14], jehož koncept byl popsán již v roce 1970 a to doktorem Coddem v článku „A relational data model for large shared data banks" [13], kde jej popisuje jako model, který je založený na operacích relační algebry. Základem je jednoduchá struktura, kdy jsou data organizovány do tabulek, které jsou složeny z řádků a sloupců. Oproti předchozím modelům dat již uživatel nemusí znát celou strukturu databáze, aby se k jednotlivým datům dostal, neboť relační model dat je založen na ukládání dat ve vztazích, které uživatel vidí jako tabulky. Nad těmito tabulkami lze provádět pomocí dotazovacího jazyku SQL operace pro práci s daty. Na obrázku 2 je zobrazeno schéma relačního modelu pro databáze objednávek.



Obrázek 2: Schéma relačního modelu dat

Každá relační databáze by měla být uživatelem chápána jako množina relací. Vztahy mezi relacemi se nazývají vazby, existují tři základní vazby:

- 1 : 1 - Jedná se o vztah, kdy je každý záznam z první tabulky svázán právě s jedním záznamem z druhé tabulky. Jedná se tedy o přímou vazbu, která by se často dala vyjádřit v rámci jedné tabulky. Jako příklad si můžeme uvést situaci z knihovny, kdy jeden čtenář může mít půjčenou právě jednu knihu a jedna kniha může být půjčena právě jednomu čtenáři. V rámci této vazby mohou nastat také situace, kdy čtenář nebude mít půjčenou žádnou knihu, nebo kniha nebude půjčená žádnému čtenáři. V tomto případě se bude jednat o vztah 1 : 0 nebo 0 : 1. S těmito vztahy je spojená i tzv. povinnost ve vztahu. Povinnost ve vztahu znamená, že v některých případech je relace ve vztahu povinná. Příkladem z knihovny může být uvedena povinnost autora u každé knihy, to znamená, že každá kniha musí mít uvedeného autora. Pokud není povinnost určená, pak jedna z relací nemusí ve vztahu figurovat. Příkladem nepovinnosti je např. existence oddělení v knihovně, které nemá žádné knihy.

- 1 : N - U tohoto vztahu je každý záznam z první tabulky svázán s `n` záznamy z druhé tabulky. Jako příklad si opět uvedeme situaci z knihovny, kdy jeden čtenář může mít půjčeno více knížek, ale jedna kniha může být půjčena právě jednomu čtenáři. Jedná se o nejvíce využívaný vztah mezi dvěma tabulkami. Největším přínosem tohoto vztahu je, že eliminuje redundanci dat na minimum. Jsou zde také zahrnuty vztahy 1 : 0, 0 : 1 a 1 : 1.

- M : N - Poslední vztah řeší případ, kdy více záznamů z jedné tabulky odpovídá více záznamům z druhé tabulky. Toto spojení se řeší pomocí rozkladu vazby M : N na vazbu 1 : N. Je vytvořena pomocná tabulka, které se říká vazební tabulka. Tato tabulka má s oběma původními tabulkami vztah právě 1 : N. Zůstaneme u příkladu z knihovny, kdy si jeden čtenář půjčí `n` knížek, ale zároveň knížka může být půjčena `n` čtenářům. V tomto případě jsou také zahrnuty vztahy 1 : 0 , 0 : 1, 1 : 1, 1 : N a N : 1.

Neméně důležité je si uvědomit, že jako záznam je brán celý řádek, který je od ostatních řádků odlišen jedinečným identifikátorem. Jak je vidět v tabulce 1, kde jsou zobrazeni uživatelé, jsou dva uživatelé téhož jména, ale odlišujeme je pomocí jejich jednoznačného identifikátoru, jimž je v našem případe login.

Login	Jméno	Příjmení
SMI102	Petr	Šmíd
SMI196	Petr	Šmíd

Tabulka 1: Tabulka uživatelů

Tabulky jsou mezi sebou propojovány identifikátory, kterým se říká klíče. Tyto klíče dělíme na primární a cizí. Primárním klíče se rozumí atribut nebo složení více atributů, které identifikují záznam dané tabulky. Cizí klíč je primární klíč jiné tabulky, a slouží jako odkaz z jednoho záznamu na záznam tabulky druhé.

2.5 Key-Value databáze

Dalším typem jsou databáze založeny na principu klíč - hodnota neboli key-value [3]. Jedná se o dvojice, kdy ke každému jednoznačnému identifikátoru je přiřazena určitá hodnota. Klíč je obdobou primárního klíče v relačních databázích ve smyslu identifikace. Pro získání dat stačí znát jen klíč, a není třeba procházet ostatní data, která jsou uložena pod hodnotou. Klíč je většinou celočíselného typu, ale existují také databáze, kde je typu textového řetězce. Jako hodnota může být číslo, textový řetězec, objekt či kolekce objektů, to už se však jedná o objektové databáze. Jelikož není vyžadováno, aby byly záznamy stejného typu, neexistuje pevné schéma dat, u kterého není potřeba prázdné hodnoty ukládat jako NULL. Typickým představitelem tohoto typu je databázový systém DynamoDB [11], jenž je využit v internetovém obchodě Amazon. Tento typ databázového modelu umožňuje značně omezenou práci s daty a to selekci, vkládání a mazání.

2.6 Objektové databáze

Objektové databáze vznikly převážné z důvodu potřeb, které relační ani jiné druhy databází nenabízejí [3]. Hlavním důvodem byla potřeba načítat velké množství záznamu za co nejkratší dobu. Cely princip je založen na objektově orientovaném programování, kdy v rámci práce s objekty, je možné přímo takovýto objekt uložit nebo jej naopak načíst z databáze. Pro uložení dat se většinou používá formát XML nebo JSON.
Existují dva druhy objektových databázi. První z nich ukládá do databáze objekt, včetně všech referencí. To znamená, že pokud například na objektu objednávka existuje reference na zákazníka, je tento zákazník uložen jako součást objektu objednávka. Tento způsob bohužel má

[11] https://aws.amazon.com/dynamodb/

velkou nevýhodu, kterou je redundance dat, protože v případě, kdy existuje deset objednávek od stejného zákazníka, jsou stejné informace o zákazníkovi jako je jméno adresa apod. uloženy desetkrát. Takové databáze jsou většinou řešeny jako list objektů. Druhým způsobem je ukládaní entit jako samostatný objekt a využívá se primárních klíčů obdobně jako je tomu u relačních databází. Výhodou oproti předchozímu případu je, že nedochází k redundanci dat. Tyto typy databází většinou používají pro uložení slovník, a jsou řešeny obdobně jako key-value. Každý objekt je tedy v databázi uložen pod svým klíčem, jímž je právě primární klíč. Pro práci s čistě objektovými databázemi se nepoužívá SQL jako je tomu u relačních objektově-relačních databází.

2.7 SQL

SQL neboli Structured Query Language je standartizovaný struktuovaný dotazovací jazyk používaný pro manipulaci s daty [10], jenž byl původně označován jako SEQUEL (Structured English Query Language), který v roce 1974 vytvořila společnost IBM[12] jako vůbec první jazyk pro práci s daty v rámci relačních databází. Jedná se o nejvíce rozšířený jazyk pro manipulaci s daty v relačních databázích. Jak se jazyk vyvíjel, vznikly různé standardy, a to ISO i ANSI, které jsou na sebe zpětně kompatibilní. V každém relačním databázovém systému jsou podporovány různé části standardu, ale z pravidla né všechny. Naopak některé systémy mají prvky, které do standardu nespadají. Poslední standardizovaná verze je SQL:2011.

Podle toho k jakým operacím se příkazy v databázi používají, je dělíme do čtyř základních skupin.

- Jazyk pro definici dat DDL (Data Definition Languag) - jedná se o příkazy, které vytvářejí či jinak upravují strukturu databáze, tabulek či atributů. Mezi hlavní příkazy této skupiny patří CREATE, ALTER, DROP...
- Jazyk pro manipulaci s daty DML (Data Manipulation Language) - jedná se o příkazy, pomocí kterých se pracuje s daty z databáze. Mezi hlavní příkazy patří SELECT, INSERT, UPDATE,DELETE...
- Příkazy pro řízení přístupu DCL (Data Control Language) - jedná se o příkazy, které se starají o uživatelské role a práva. Mezi hlavní příkazy GRANT,REVOKE...
- Příkazy pro řízení transakcí TCC (Transaction Control Commands) - jedná se o příkazy, které obstarávají práci s transakcemi. Mezi hlavní příkazy patří BEGIN, COMMIT, ROLLBACK...

[12] https://www.ibm.com/

2.8 Transakce

Pojmem transakce se v databázích označuje skupina operací, které převádějí databázi z jednoho konzistentního stavu do druhého [9]. Počet ani rozsah operací, které jsou v rámci jedné transakce provedeny, není jakkoli omezen. Zabalení více operací do transakce může rapidně snížit dobu jejich zpracování. Podle způsobu zpracování transakce je rozlišujeme na tzv. pesimistické a optimistické zpracování. Pesimistické zpracování zaznamenává změny do dočasných tabulek, které jsou platné po dobu transakce. Po potvrzení transakce dojde k aktualizaci, kdy jsou data z dočasných tabulek převedena do stálých. U optimistického zpracování se naopak předpokládá, že při provádění transakce nedojde k chybě, a proto jsou data rovnou ukládána do stálých tabulek. Pokud by došlo k chybě, existují tři základní techniky zotavení.

- První je odložená aktualizace, kdy během transakce nedochází k okamžité aktualizaci databáze ani logu, až do doby, kdy je transakce potvrzena. Data jsou během této transakce uložena v paměti. Pokud by došlo k chybě během zpracování transakce, není zapotřebí provádět žádné úkony pro obnovení dat.
- Druhou technikou je okamžitá aktualizace, kdy jsou během transakce data aktualizována přímo v databázi. Původní data jsou zálohována v logu. V případě selhání transakce dojde k obnovení původních dat právě z této zálohy.
- Třetím způsobem zotavení je kombinovaná technika předchozích dvou bodů, kdy k aktualizaci databáze dochází v určitých časových intervalech, které se nazývají kontrolní body.

Pro práci s transakcemi jsou používány tyto základní příklady:

- *COMMIT* - Potvrzení transakce a uložení dočasných výsledků do databáze.
- *ROLLBACK* - Odvolání změn, návrat do stavu před započetím vykonávání transakce.
- *BEGIN TRAN* - Začátek transakce .

2.9 ACID

Jedná se o skupinu vlastností, které by měla každá transakce splňovat, aby nedošlo k nekonzistenci dat. Tato zkratka je složením 4 vlastností, a to:

- *Atomičnost* - Každá transakce musí být vždy nedělitelná a musí být provedena jako jeden celý celek. To znamená, že jsou buď provedeny všechny operace transakce, anebo žádná.
- *Konzistence* - Při a po vykonání transakce musí být vždy data konzistentní a nesmí dojít k porušení integritních omezení. Pokud dojde k selhání transakce, nesmí být žádná operace uložena.
- *Izolovanost* - Transakce jsou izolované od okolí, a všechno co je uvnitř transakce je pro ostatní transakce neviditelné.
- *Trvalost* - Po úspěšném průběhu transakce jsou provedené změny uloženy v databázi natrvalo. Je zaručeno, že nedojde ke ztrátě nebo poškození dat v důsledku selhání aplikace, nebo databázového systému.

2.10 Embedded databázové systémy

Zvláštní skupinu databázových systémů tvoří embedded databázové systémy, které jsou řešeny jako knihovny, které se přilinkují k aplikaci a využívají jejího adresného prostoru [4]. Tyto databázové systémy jsou primárně určeny pro práci v single-user režimu. To znamená, že nepodporují víceuživatelský přístup a jsou ideální pro malé a středně velké aplikace.

Ačkoliv se může zdát, že se jedná o nové databázové systémy opak je pravdou. Za dob, kdy se jako operační systém výhradně používal MS-DOS a počítačové sítě téměř neexistovaly, nebylo třeba víceuživatelského přístupu. Většina aplikací tak pracovala v single-user režimu. Jednou z mnoha výhod těchto databázových systémů je určitě jejich jednoduchost. Nenabízejí sice takové možnosti jako serverové databázové systémy, ale v aplikacích, jenž tyto embedded systémy využívají to většinou není třeba. Nespornou výhodou je velikost těchto systémů, jelikož přilinkování těchto systémů databázi o moc nezvětší. V provozu těchto databází existují desítky již odladěných knihoven, a tak nehrozí potenciální problém s řešením chyb jako při vlastním vývoji. Pro provoz aplikací je většinou třeba mít zvlášť nainstalovaný software pro podporu databázových systému. Tohle ovšem u embedded databázových systému neplatí. Většinou jsou dynamicky přilinkovány a jsou instalovány společně s aplikací, a tak se uživatel nemusí starat o instalaci či nastavení. Provoz těchto databázových systémů je většinou bezstarostný, na rozdíl od serverových, u kterých je potřebná správa.

Co naopak u těchto systému se však nenajde, jsou třeba přístupová práva. Vzhledem k tomu, že pracují v single-user režimu tak to ani není třeba. Aby byl zajištěn současný přístup z více

aplikací, jsou v embedded databázových systémech implementovány také možnosti zamykání. Nejsou však tak sofistikované, jako je tomu u serverových databází. Většinou se jedná o dva druhy, a to readonly a read-write, což zajistí, že z databáze může současně číst několik aplikací, ale pouze jedna zapisovat. Většina embedded databázových systému nevyužívá SQL až na výjimky (např. SQLite), ale naopak využívá objektového modelu s typem Key-value. Embedded databázové systémy podporují základní práci s daty, a to selekci, mazání, vkládání a aktualizaci.

Tyto systémy přinášejí bezesporu velké uplatnění v mobilních zařízeních. Pro jejich implementaci je možné využít velkou škálu programovacích jazyků a prostředí. Mezi nejznámější zástupce embedded databázových systému pro mobilní zařízení patří SQLite, RealmDB, LiteDB[13], což je open source řešení vycházející z MongoDB[14], dále CouchBaseLite, Interbasse[15], LevelDB[16] a mnoho dalších.

2.11 Benchmarking

V oblasti IT se pod pojmem benchmarking označuje výkonnostní měření systému za pomocí programů, funkcí či operací, které lze kategorizovat, srovnávat či hodnotit podle předem stanovených hodnot [15]. Pojem benchmarking se většinou spojuje s testováním různých komponent hardwaru, jako je například výkon procesoru pomocí operací s plovoucí desetinnou čárkou. Nejedná se však čistě o záležitost hardwaru, nýbrž také softwaru, jako například výkonnostní testování databází. Na internetu existují desítky hotových benchmarkingových nástrojů, žádný z nich však není uznán jako nějaký standard. V oblasti testování databázových systémů existuje TPC[17] (Transaction Processing Performance Council), což je nezisková společnost, která se zabývá právě testováním různých počítačových systémů. Jejich nejznámější benchmark je TPC-C, který měří a porovnává výkony OLTP (online transaction processing).

Návrh a implementace benchmarku je velmi složitý proces, u kterého by mělo být dodrženo pět základních vlastností.

- *Relevantnost* - Testovaná data by měla pocházet z reálného světa, což znamená, že návrh testu by měl vycházet ze situace, která reálně nastane v praxi.
- *Opakovatelnost* - Každý test by měl být kdykoliv opakovatelně spustitelný a získané výsledky by se měly vždy shodovat.
- *Spravedlivost* - Každá z testovaných jednotek by měla mít stejný podíl na implementaci benchmarku.

[13] http://www.litedb.org/
[14] https://www.mongodb.com/
[15] https://www.embarcadero.com/products/interbase
[16] http://leveldb.org/
[17] http://www.tpc.org/

- *Ověřitelnost* - Výsledky testování musí být vždy pravdivé a nesmí dojít k publikování zkreslených či nepravdivých výsledků.
- *Ekonomičnost* - Cena vývoje i samotného testování by neměla být dražší než vývoj testované aplikace.

Nejčastějším problémem je dodržení všech pěti vlastností, a tak tvůrce benchamarku musí od některé upustit nebo jí omezit. V případě první a poslední vlastnosti lze stanovit, že se prakticky vylučují. Pro správné otestování databázového systému, který se stará o objednávky zákazníků, je zapotřebí nejprve připravit aplikační vrstvu, která by byla velmi podobná té reálné a jejíž vývoj by něco stál. V rámci této práce není využito žádného již hotového benchmarku, neboť existující benchmarky jsou zaměřeny na různé druhy databázových systémů nebo porovnávají databázové systémy jen z pohledu vkládání a čtení dat. Detailnímu popisu připraveného benchmarku je věnována samostatná kapitola.

3 Vybrané databázové systémy pro testování

V rámci testování jsou vybrány a popsány následující čtyři embedded databázové systémy pro mobilní zařízení, které mají podporu pro vývoj pomocí Xamarin[18], a to podle pořadí v rankingu z internetového portálu db-engines.com[19] a podle počtu použití knihoven z NuGet[20], jenž je distribuční platforma pro knihovny a komponenty všeho druhu. Princip vyhodnocování pořadí na internetovém portálu db-engines.com je založen na popularitě jednotlivých databázových systémů.
Mezi hlavní kritéria patří:

- Počet zmínek o databázovém systému na internetových stránkách Google[21], Yandex[22] a Bing[23].
- Frekvence vyhledávání přes vyhledávač Google.
- Počet diskuzí o databázovém systému na poprtálech Stack Overflow[24] a DBA Stack Exchange[25].
- Počet pracovních nabídek, ve kterých je vyžadován daný databázový systém na portálech Indeed[26] a Simply Hired[27].
- Počet profilů, u kterých je jako dovednost uveden daný databázový systém na portálech LinkedIn[28] a Upwork[29].
- Počet tweetů na sociální síti Twitter.

3.1 SQLite

SQLite je relační databázový systém vyvinutý v roce 2000 D. Richardem Hippem. Jedná se o systém šířený pod licencí public domain, což znamená, že je volně šiřitelný a nepodléhá ochraně autorských práv [5]. I když se jedná o čistě relační embedded databázový systém, není zde možnost jej konfigurovat v takové míře, jako je tomu u serverových databázových systémů. Určité možnosti tu však jsou, a to pomocí příkazu PRAGMA. Velkou výhodou systému SQLite

[18] https://www.xamarin.com/
[19] https://db-engines.com/en/ranking
[20] https://www.nuget.org/
[21] https://www.google.com/
[22] https://yandex.com/
[23] https://www.bing.com/
[24] https://stackoverflow.com/
[25] https://dba.stackexchange.com/
[26] https://indeed.com
[27] https://www.simplyhired.com/
[28] https://www.linkedin.com/
[29] https://www.upwork.com/

je, že z velké části podporuje celý standard SQL-92. SQLite podporuje dokonce i cizí klíče, jejichž vynucení je však potřeba aktivovat použitím již zmíněného příkazu PRAGMA.

Bohužel SQLite každý vkládaný záznam vkládá jako jednu transakci, což značně snižuje výkon tohoto systému. Toto se však dá omezit obalením většího množství insertů do jedné transakce pomocí příkazů BEGIN TRAN a COMMIT stejně jako u serverových databází. Další výhodou je plně podporovaný ACID. Jelikož se jedná o embedded databázový systém, chybí zde věci jako je víceuživatelský přístup, přístup z více vláken či řešení přístupových práv. Asi největším problémem u tohoto systému je, že zde chybí možnost synchronizace dat se serverovým či cloudovým databázovým systémem. Synchronizace dat se většinou u aplikací, které pro práci s daty používají systém SQLite řeší pomocí načtení dat do paměti zařízení, a poté jsou aplikační vrstvou uloženy do jiného databázového systému. Pokud je třeba s aplikací pracovat síťově či data synchronizovat přes cloud, není SQLite ideálním řešením. Pro SQLite je určitě specifické to, že nemá pevně danou strukturu. Místo ní je použita tzv. typová afinita, což znamená, že do každého sloupce lze zadat jakýkoliv datový typ, který je následně převeden na typ sloupce, pokud je to možné nebo ponechána tak, jak je. Kromě podpory transakcí a možnosti uzamknout soubor, kde jsou uloženy data, podporuje SQLite také triggery. Uložené procedury zde však nenajdete, jelikož je databázový systém přímo nalinkován do aplikace, tak to ani není potřebné.

Podpora programovacích jazyků, technické specifikace či počty použití knihoven, jsou zobrazeny v tabulce 2 na konci této kapitoly. Mezi nejvýznamnější odběratele tohoto systému patří Skype[30], Google[31],nebo také antivirový program McAfee[32].

3.2 RealmDB

RealmDB je dokumentově orientovaný embedded databázový systém určený pro mobilní zařízení podporující Android a iOS [6]. Vývoj produktu začal již v roce 2010, a to pod jménem TightDB. V roce 2014 byl TightDB přejmenován na RealmDB a byl vydán k testování pro veřejnost. První stabilní verze tohoto systému byla vydána v lednu roku 2017. Jedná se o systém licencovaný pod licencí Apache license, což znamená, že je po uživateli vyžadováno zachování autorství a zřeknutí se odpovědnosti. Tento druh licencování umožňuje uživatelům používat software k nejrůznějším účelům jako jsou například distribuce, úpravy či redistribuce upravených verzí. Jako definice schémat jsou použity přímo třídy modelu. Vazby typu 1 : N jsou řešeny jako odkazy na objekty, podobně jako vztahy M : N, které se dají řešit například pomocí kolekcí typu list. V každé třídě, která je mapována do databáze lze nastavit jeden primární klíč. Primární klíč může být typu znak, řetězec nebo integer a nelze jej již měnit. Další důležitou vlastností je možnost indexovat jednotlivé atributy. Mezi podporované atributy patří řetězec, integer, bool a datetime. Každá operace prováděná nad databází je řešena jako samostatná transakce, ale je zde také možnost obalit více operací do jedné transakce, čímž se zkrátí doba provádění operací. Jelikož RealmDB

[30] https://www.skype.com/cs/
[31] https://www.google.com/
[32] https://www.mcafee.com/cz/index.html

nepodporuje hromadné operace jako je například bulk insert, je u většího množství vkládaných dat obalení do transakce jedinou možností, jak dobu vkládání dat urychlit. Pro dotazy není podporován SQL, ale využívá se standardů LINQ syntaxe včetně možností třídění. Na rozdíl od SQLite má RealmDB i svou serverovou část pod název Realm Sync, jež umožňuje synchronizaci dat mezi mobilními zařízeními a serverem. Bohužel serverová část nespadá pod volné licence a uživatel si jí v případě potřeb synchronizace musí koupit. Z toho důvodu není synchronizace v rámci této práce testována. RealmDB má ve světě opravdu velké zastoupení, jež se svými dvěma miliardami instalací se jedná o jeden z největších embedded databázových systémů. Mezi největší odběratelé patří firmy Amazon[33], McDonald[34], Nike[35], BBC[36], Starbucks[37], Adidas[38], Loreal[39], CISCO[40], Google či Ebay[41].

3.3 CouchBaseLite

CouchBaseLite je stejně jako RealmDB dokumentově orientovaný embedded databázový systém určený speciálně pro mobilní zařízení, který ukládá data v JSON formátu [7]. JSON neboli JavaScript Object Notation je univerzální formát pro ukládání dat, která mohou být organizována v polích nebo ukládána jako objekty. Jedná se o alternativu k více známému formátu XML. Výhodou tohoto formátu je jeho nezávislost na jakékoliv platformě a jednoduchost, neboť je čitelný i pro člověka. CouchBaseLite stejně jako RealmDB nepodporuje SQL, ale využívá vlastní jazyk, a to N1QL, což je alternativa k SQL určená pro práci s dokumenty typu JSON, pomocí kterého je možné používat příkazy pro selekci, vkládání, mazání a aktualizaci dat. Systém CouchBaseLite pochází z rodiny CouchBase a všechny jejich produkty jsou open source. Mezi další produkty patří CouchBase Server, který je také založen na formátu JSON, ale na rozdíl od verze Lite se jedná o klasickou architekturu klient server. Dalším produktem z rodiny CouchBase je Mobile 2.0, který byl vydán v dubnu 2018, jedná se o rozšiřující verzi pro verzi Lite, který je doplněn o možnost synchronizace s již představeným produktem CouchBase Server, a to za pomoci posledního produktu, kterým je SyncGateway, což je služba obstarávající komunikaci mezi mobilní a serverovou verzí. V rámci testování se budeme primárně věnovat produktu Lite, neboť v době zahájení této práce ještě nebyla verze Mobile uvolněna.

Dokumenty jsou uloženy principem key-value. Stejně jako tomu bylo u předchozích systémů, i zde je možnost indexovat jednotlivé atributy, ale je třeba si dávat pozor, aby v jednom objektu nebylo indexů stejně, jako je atributů, a to z důvodu velkého zatížení a snížení výkonu. CouchBaseLite podporuje také fultextové vyhledávání, což kolikrát dokáže zjednodušit práci s daty,

[33] https://www.amazon.com/
[34] https://www.mcdonalds.cz/
[35] https://www.nike.com
[36] http://www.bbc.com/
[37] https://www.starbucks.com/
[38] https://www.adidas.com/
[39] https://www.loreal.com/
[40] https://www.cisco.com
[41] https://www.ebay.com/

je ale důležité prvně si vytvořit fultextový index. Všechny operace, které jsou rozváděny jako transakce, kterou si řídí systém sám, to znamená, že není možné uživatelsky obalit více operací do jedné transakce jako je tomu u předchozích systémů. CouchBase je významným hráčem na trhu s embedded databázovými systémy, čehož jsou důkazem zákazníci jako je LinkedIn[42], Ebay, Cisco, Skype, Viber[43] a mnoho dalších.

3.4 LiteDB

Posledním systémem je LiteDB, který je také dokumentově orientovaný embedded databázový systém [8]. Jedná se o projekt, který vznikl v roce 2014 a má za cíl vytvoření zjednodušené verze databázového systému MongoDB pro mobilní zařízení. Stejně jako předešlý systém je i tento systém open source. Objekty jsou ukládány do databáze ve formátu BSON, což je binární verze formátu JSON. Výhodou je dynamické databázové schéma, které umožňuje v jednom dokumentu ukládat různé objekty. Stejně jako je tomu u relačních databází, je i zde možnost indexovat všechny atributy a to včetně sekundárních indexů. Indexy jsou v LiteDB řešeny pomocí skip listu, který je alternativou k vyváženým stromům. Pro manipulaci s daty je využito standardů LINQ obdobně jako je tomu u RealmDB. Jako jeden z mála embedded databázových systému splňuje vlastnosti ACID, z čehož plyne, že operace jsou prováděny jako transakce. Jde o single user databázi, čímž není možné k dokumentu přistupovat z více vláken. Obdobně jako je tomu u předešlého systému, není potřeba žádná konfigurace a stačí pouze připojit knihovnu. Na rozdíl od ostatních testovaných systému se jedná o poměrně neznámý systém a pro testování byl vybrán hlavně z důvodu zájmu o databázový systém MongoDB[44], který bohužel nedisponuje embedded verzí pro mobilní zařízení.

[42] https://www.linkedin.com/
[43] https://www.viber.com/
[44] https://www.mongodb.com/

3.5 Srovnání

V tabulce 2 je porovnání vybraných databázových systému z mnoha různých pohledů a samotných vlastností.

	SQLite	**CouchBaseLite**	**RealmDB**	**LiteDB**
Datový mode	Relační	Objektový	Objektový	Objektový
Podpora transakcí	ANO	ANO	NE	ANO
ACID	ANO	NE	NE	ANO
Podpora indexů	ANO	ANO	ANO	ANO
Jazyk pro manipulaci s daty	SQL-92	N1QL	LINQ	LINQ
Licence	Public domain	Apache License 2.0	Apache License 2.0	MIT License
Možnost synchronizace	NE	ANO	Ano - komerční	NE
Formát ukládaných dat	-	JSON	JSON	BSON
Podpora iOS	ANO	ANO	ANO	ANO
Podpora Android	ANO	ANO	ANO	ANO
Pořadí dle db-engines.com	11.	24.	50.	188.
Počet stažení NuGet	765 000	33 000	145 000	235 000
BULK operace	ANO	ANO	NE	ANO
Jazyk implementace	C	C++	C	C#
Více uživatelský přístup	NE	NE	NE	NE
Zámky	ANO	ANO	ANO	ANO
Velikost knihovny	500kB	500kB	1MB	350kB

Tabulka 2: Tabulka srovnání vybraných systémů

4 Popis benchamarku a testovací aplikace

K testování není využito žádného z již hotových benchmarků, ale pro potřeby testování je navržen vlastní benchmark. V této kapitole bude popsán celý princip testování vybraných databázových systémů, včetně představení vybraného databázového schématu a detailní popis samotné aplikace vytvořené k testování.

4.1 Teoretický popis

Testování je založeno na objednávkovém systému, a je složeno z celkem tří testů, které jsou prováděny nad různými množinami dat, které jsou zobrazeny v tabulce 3. V rámci každého testu bude testováno celkem pět operací a to vložení daného počtu záznamů, načtení záznamu dle identifikátoru z daného počtu záznamů, načtení daného počtu záznamů, aktualizace daného počtu záznamů a mazání daného počtu záznamů.

Testovaná operace	**počet záznamů v 1. testu**	**počet záznamů v 2. testu**	**počet záznamů v 3. testu**
Vkládání	100	1 000	50 000
Aktualizace	100	1 000	50 000
Hromadné vyhledávání	100	1 000	50 000
Vyhledání jednoho záznamu	100	1 000	50 000
Mazání	100	1 000	50 000

Tabulka 3: Tabulka objemu kolekcí pro testování

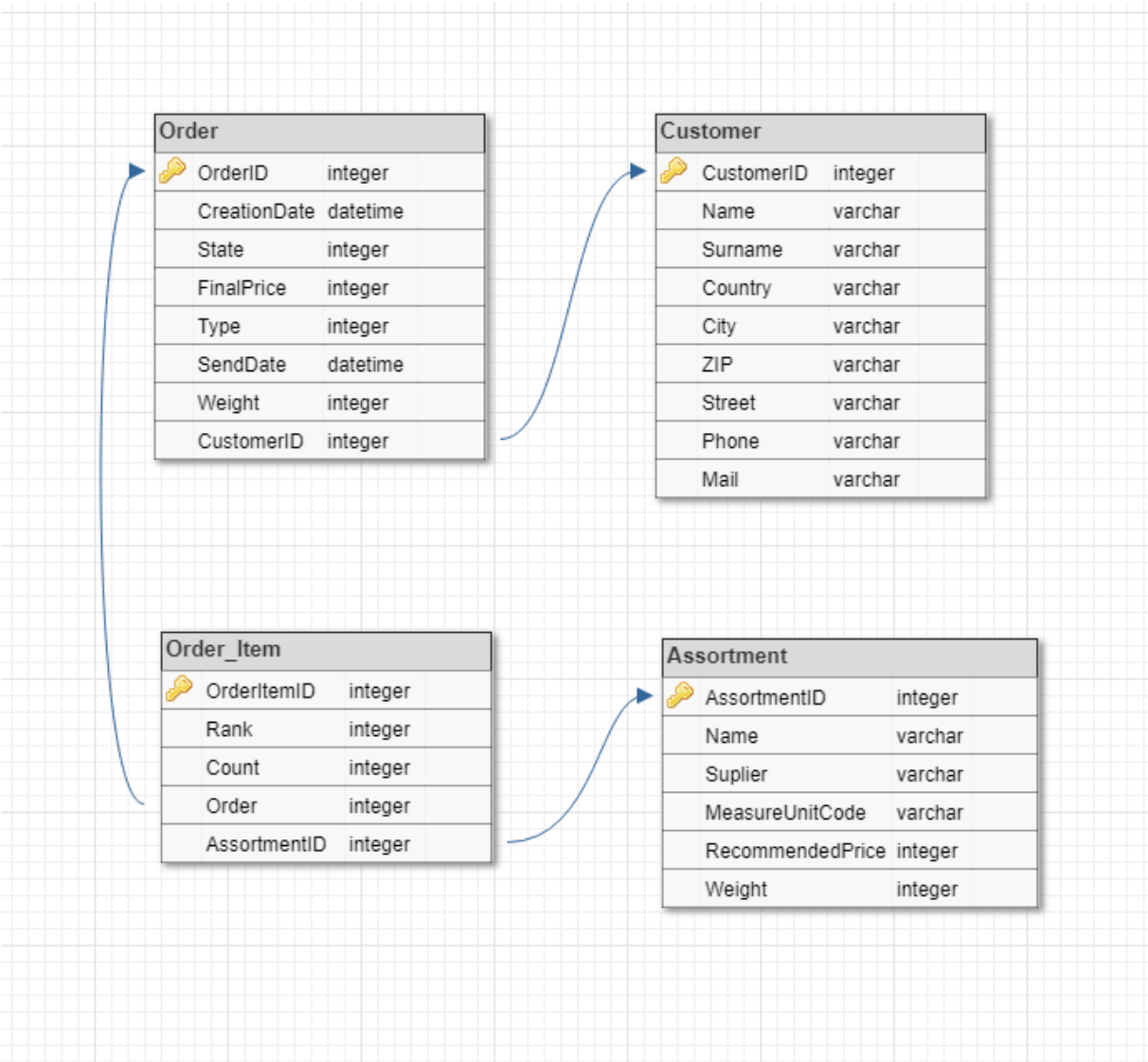
V případě relačních databází jsou objednávky načítány pomocí příkazu JOIN. Počty dat v kolekci jsou navrženy dle potřeb mobilních aplikací od malých firem, kdy se pracuje se stovkami záznamů, až po velké servisní střediska, která denně odbaví desítky tisíc objednávek. Pro větší kolekce dat nejsou embedded databázové systémy optimální neboť jejich výkon při vysokých počtech zaostává. V rámci každého testu je měřen čas v milisekundách potřebný na zpracování daných operací. Na začátku každého testu je databáze prázdná a dochází k vytvoření datové struktury, následně je vloženo `n` záznamů, kdy `n` odpovídá testované množině dat. Záznamem je vždy myšlena objednávka s položkami a se zákazníkem. Po vložení záznamů dochází k zajištění času potřebnému pro načtení kolekce dat o `n` záznamech, kdy `n` odpovídá testované množině dat. Třetí část každého testu je tvořena přeceněním objednávek, čili k aktualizaci `n` záznamů v databázi, kdy `n` opět odpovídá testované množině dat. Čtvrtým krokem je načtení jedné objednávky dle čísla objednávky, jenž je primárním klíčem a je nad ní v každém systému založen index. Posledním krokem je smazání `n` objednávek, kdy `n` odpovídá počtu testované kolekce dat. Test má demonstrovat denní koloběh firmy zpracující objednávky. Na začátku dne je do mobilního zařízení vložen určitý počet objednávek k vyřízení. Během dne dojde k vyřízení všech objednávek a k jejich aktualizaci v zařízení. V případě potřeb má pracovník možnost zobrazit seznam všech objednávek včetně všech informací s nimi spojenými nebo zobrazit vybranou objednávku dle čísla objednávky. Na konci dne dojde ke smazání již vyřízených objednávek. V rámci testů není implementována synchronizace dat mezi mobilním zařízením a jiným typem

databázového systému z důvodu, že by nedocházelo k testování vybraných systémů, ale k testování aplikační vrstvy, která by tuto funkčnost zajišťovala. Každý test je celkově proveden 10×, a v rámci výsledků je vyhodnocen průměr ze všech deseti testů.

4.2 Popis databázového modelu

Jelikož databázový model pro jednotlivé databázové systémy je odlišný, jsou popsány odlišné typy samostatně.

- SQLite - relační model dat - databázový model je složen z celkem čtyř tabulek. První tabulkou je objednávka, která obsahuje číslo objednávky, a je primárním klíčem. Dalším atributem je datum vytvoření objednávky a stav objednávky, který může nabývat hodnot "založená objednávka", "vyřízená objednávka"a "stornována objednávka". Následuje celková cena objednávky a typ objednávky, jež zde může být typu "online"nebo "osobní objednání". Mezi posledními atributy patří datum odeslání objednávky, celková hmotnost objednávky a identifikační číslo zákazníka, jenž je cizím klíčem z tabulky zákazník. Druhou tabulkou je zákazník, jenž má vazbu na objednávku a je ve vztahu 1 : N, kdy jeden zákazník může mít více objednávek. Tabulka zákazník obsahuje celkem devět atributů, a to identifikační číslo zákazníka, jenž je primárním klíčem, dále jméno a příjmení, stát, město, adresu, poštovní směrovací číslo, telefon a mail. Třetí tabulkou je položka objednávky mající opět vazbu na objednávku 1 : N , kdy každá objednávka může mít několik položek. Každá položka má pět atributů a to identifikační číslo položky, pořadí položky na objednávce, počet kusů sortimentů, identifikační číslo objednávky, jenž je cizím klíčem z tabulky objednávka a identifikační číslo sortimentu, které je cizím klíčem z tabulky sortiment. Poslední tabulkou je tabulka sortimentů, která má vazbu na tabulku položka sortimentu 1:N, kdy každý sortiment může být na několika položkách objednávky. Na obrázku 3 je zobrazeno schéma.



Obrázek 3: Relační schéma

- RealmDB, CouchBaseLite, LiteDB - objektový model dat - databázový model je složen z jednoho objektu, a to objednávka, který obsahuje všechny informace jak o objednávce, tak položky objednávky, informace o zákazníkovi i podrobnosti o sortimentu.

4.3 Analýza

Pro potřeby testování je třeba vytvořit aplikaci, pomocí které bude možno otestovat vybrané databázové systémy a to pro operace selekce, vkládání, aktualizace a mazání záznamů z pohledu délky vykonávání dané operace. Aplikace bude pracovat se čtyřmi objekty a to objednávka, položka objednávky, zákazník a sortiment. Pro potřeby testování není třeba, aby aplikace vizuálně zobrazovala data, se kterými bude pracovat. Po vykonání testované operace, bude na display zařízení zobrazen čas v milisekundách potřebný pro její vykonání. Klíčové funkce aplikace, které je třeba implementovat, jsou generace předem daných kolekcí dat o různých objemech, metody pro měření času potřebného pro vykonání testovaných operací a samotné metody pro testované operace s vybraným databázovým systémem. U generovaných dat není třeba zajistit jedinečnost atributů, výjimkou jsou samozřejmě primární klíče. Z důvodu specifických potřeb jednotlivých databázových systémů, bude pro každý testovaný systém vytvořena jedna aplikace, která bude pracovat vždy se stejnou kolekcí dat. Výsledná aplikace bude spustitelná na mobilním telefonu podporující operační systém Android.

4.4 Použité technologie

Výsledná aplikace je vytvořena v prostředí Xamarin[45] ve Visual studiu 2017, které umožňuje vývoj aplikací pro mobilní zařízení podporující operační systém Android[46]. Prezentační vrstva je napsána ve značkovacím jazyce XAML, který je velice podobný jazyku HTML. Aplikační vrstva je napsána za pomocí frameworku .NET a to v jazyce C sharp. Jednotlivé knihovny embedded databázových systémů budou do aplikace připojeny z distribuční platformy NuGet[47].

4.5 Aplikační vrstva

Aplikační vrstva je složena ze dvou částí. První části jsou modely pro jednotlivé objekty objednávek, položek objednávky, sortimentů a zákazníků, které jsou ve složce *Models* tvořeny třídami *Order.cs*, *OrderItems.cs*, *Assortment.cs* a *Customer.cs*. Modely jsou u každé aplikace jiné, a to z důvodu potřeby definovat různé vlastnosti tříd a atributů u jednotlivých databázových systémů. U systému SQLite je to definice primárních a cizích klíčů. V případě systému RealmDB je třeba, aby tyto modely dědily z třídy RealmObject, která se stará o mapování objektů. U systému LiteDB a CouchBaseLite není třeba specifické implementace.
Druhou části je třída *OrderViewModel.cs*, která se nachází ve složce *ViewModels*, a se stará o generaci a práci s daty, práci s databázovým systémem a samotné testování. Princip generace dat je společný pro všechny vytvořené aplikace. Vkládaná data jsou vytvořena vždy se stejnými obsahy, a liší se pouze v identifikátorech. Počet vytvářených dat je zadán uživatelem. V rámci všech aplikací jsou generované data uloženy do kolekce, se kterou aplikace nadále pracuje.

[45] https://www.xamarin.com/
[46] https://www.android.com/
[47] https://www.nuget.org/

Pro stisknutí jednotlivých tlačítek jsou připraveny příkazy, které volají metody vytvořené pro testování jednotlivých operací.

- Vložení dat - prvním krokem je vytvoření kolekce o zadaném množství dat, následující spuštěním měření času a vložením dat do databáze pomocí metod specifických pro daný databázový systém. Po vložení dat do databáze dochází k zastavení měření času a jeho zobrazení.
- Hromadná selekce - první krokem je spuštění měření času, následující načtením dat z databáze a uložením do vytvořené kolekce. Po načtení dat do kolekce dochází k zastavení meření času a jeho zobrazení.
- Selekce jednoho záznamu - první krokem je spuštění měření času, následující načtením jednoho záznamu dle identifikačního čísla zadaného uživatelem. Po načtení záznamu dochází k zastavení měření času a jeho zobrazení.
- Aktualizace - první krokem je spuštění měření času, následující uložením aktualizovaných záznamů do databáze. Po uložení aktualizovaných dat do databáze dochází k zastavení měření času a jeho zobrazení.
- Mazání - první krokem je spuštění měření času, následující smazáním dat v databázi. Po smazání dat v databázi dochází k zastavení měření času a jeho zobrazení.

Z důvodu, že práce s daty je u jednotlivých databázových systémů odlišná, je vysvětleno, jakým způsobem aplikace s daty pracuje v daném databázovém systému

4.5.1 Návrh aplikace s využitím systému SQLite

Pro systém SQLite je nejprve nutno vytvořit spojení pomocí metody SQLiteConnection, která má dva parametry a to cestu a název souboru, ve kterém jsou data uložena [5]. Po vytvoření připojení je možné používat všechny metody v knihovnách databázového serveru SQLite. Následuje vytvoření tabulek, které jsou vytvořeny automaticky pomocí metody CreateTable, která tabulku vytváří dle typu objektu. Pro metodu vkládání dat do databáze je možnost využití dvou metod a to pro vkládání jednotlivých záznamů, nebo pro vložení celé kolekce. V případě aplikace se jedná o kolekci typu seznam. Metody pro mazání dat umožňují smazat buď jednotlivé záznamy, nebo obsah celé tabulky. Aktualizace dat je řešena stejně jako vkládání dat a to aktualizací jednoho záznamu nebo celé kolekce. Pro načtení dat z databáze je možné využit několika metod. V případě testovací aplikace jsou data načteny pomocí metody Table, která vrací obsah celé tabulky, nebo je možné využit jazyka SQL.

4.5.2 Návrh aplikace s využitím systému LiteDB

Nejjednodušší je práce se systémem LiteDB, obdobně jako u systému SQLite stačí pro práci s databází vytvořit spojení a to pomocí metody LiteDatabase s jedním parametrem a to souborem

s databází [8]. Na rozdíl od SQLite není potřeba řešit existenci databázové struktury, a je možné hned využit příkazu knihoven LiteDB. Vkládání dat do databáze je řešeno metodami Insert pro vložení jednoho záznamu a InsertBulk pro vložení kolekce záznamů. Mazání je řešeno metodou delete, kdy mazané data jsou vybrány pomocí LINQ. Při aktualizaci dat je nutné nejprve objekt z databáze načíst, poté jej aktualizovat a následně pomocí metody Update aktualizovat v databázi. Načtení dat z databáze je řešeno metodou FindByID pro načtení jednoho záznamu nebo metodou FindAll pro vrácení kolekce záznamů.

4.5.3 Návrh aplikace s využitím systému RealmDB

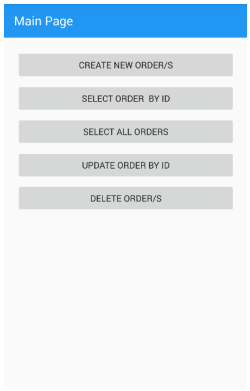
Systém RealmDB pro práci s daty vyžaduje nejprve vytvoření instance bezparametrovou metodou GetInstance, pomocí které se dále přistupuje k metodám definovaných v knihovnách RealmDB [6]. Pokud neexistuje soubor, kde jsou data ukládány, dojde k jeho automatickému vytvoření, o tuto část se v tomto případě stará pouze RealmDB. Před prací se souborem, kde jsou data uložena je nutno otevřít spojení pomocí BeginWrite, čímž dojde také k zahájení transakce. Následně lze pomocí metody Add přidávat objekty do databáze. Pro přidání kolekce objektů je nutno celou kolekci projít a jednotlivé objekty uložit postupně. Mazání se řeší metodami Remove, kdy je mazán vybraný objekt, nebo RemoveAll, kdy je smazána celá databáze. Metodou Add lze také data aktualizovat a to přidáním parametru update, který má hodnotu true. Pro selekci dat slouží metoda All, která vrací celý obsah databáze. Pro výběr záznamu dle různých atributů se využívá LINQ.

4.5.4 Návrh aplikace s využitím systému CouchBaseLite

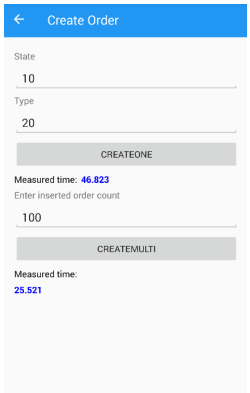
Pro práci s daty u systému CouchBaseLite, je nutno nejprve vytvořit tzv. manažera, pomocí kterého pracujeme s databázovým systémem CouchBaseLite [7]. Následně je pomocí manažera vytvořena instance, kterou lze pracovat s vybraných souborem. Ve vybrané instanci je nutno založit dokument, a nastaviv jeho identifikátor. Pokud dokument neexistuje je vytvořen pomocí metody CreateDocument. Nad dokumentem lze dále provádět metody obsažené v knihovnách CouchBaseLite. Na rozdíl od ostatních systémů, se nepoužívá pro data kolekce seznam ale slovník. Pro operace vkládání a aktualizace je využito metody PutProperties, kdy jsou nejprve do slovníku načteny původní data z databáze, a následně jsou vloženy aktualizované záznamy do databáze. Pro výběr dat z databáze je použito metody Properties. Mazání dat je řešeno metodou Remove pro jeden záznam nebo smazáním celého dokumentu.

4.6 Prezentační vrstva

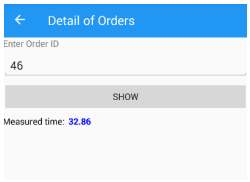
Prezentační vrstva je složena ze souborů ve složce *Views*, kde pro každou obrazovku aplikace existují 2 soubory a to *.xaml a *.xaml.cs. soubory s příponou .xaml se starají o grafické rozložení jednotlivých komponent a bindování příkazů, uložených ve třídě *OrderViewModel.cs*, a soubory s příponou xaml.cs, které jsou automaticky vytvořeny v projektu, a nejsou nijak editovány. Na obrázku 4 je vidět domovská obrazovka, z které lze pomocí jednotlivých tlačítek přejít do obrazovek pro vybraný test. Jelikož se jedná o navigační menu, lze se kdykoliv vrátit na tuto obrazovku stisknutím tlačítka zpět. Pro jednotlivé testy je vždy vytvořena samostatná obrazovka. Na obrázku 5 je zobrazena obrazovka sloužící pro generaci a následné vkládání záznamů do databáze. Je zde vidět tlačítka pro vložení jednoho nebo více záznamů, textové pole pro zadání stavu, typu a počtu objednávek a naměřené časy potřebné pro vložení jedné a sta objednávek. Další obrazovka zobrazena na obrázku 6 slouží pro výběr jedné objednávky dle zadaného identifikačního čísla.



Obrázek 4: Úvodní obrazovka

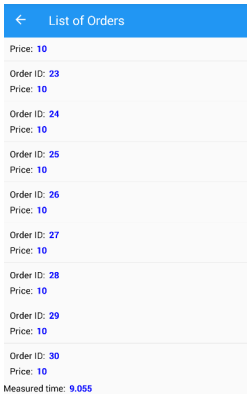


Obrázek 5: Obrazovka pro vkládání záznamů

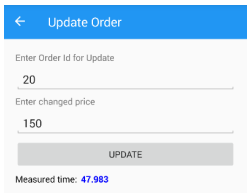


Obrázek 6: Obrazovka pro načtení záznamu dle ID

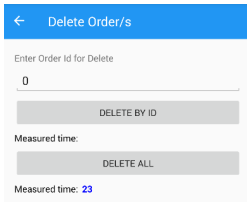
Na obrázku 7 je obrazovka, ve které jsou zobrazeny identifikační čísla a ceny všech objednávek z databáze. Tyto objednávky jsou načteny při otevření této obrazovky, stejně jako čas potřebný pro jejich načtení. Pro testování aktualizací záznamů je zobrazena obrazovka na obrázku 8. Zde je vidět dvě textové pole, první slouží pro zadání počtu záznamů, které budou aktualizovány a druhé pro cenu, která bude na aktualizovaném záznamu změněna. Poslední obrazovka je určena pro mazání záznamů a je zobrazena na obrázku 9



Obrázek 7: Obrazovka pro hromadné načtení záznamů



Obrázek 8: Obrazovka pro aktualizace záznamů



Obrázek 9: Obrazovka pro pro smazání záznamů

5 Výsledky testování

V rámci této kapitoly jsou všechny testované části vyhodnoceny jednotlivě za pomocí názorných tabulek a grafů. Na závěr kapitoly je celkové vyhodnocení jednotlivých systémů.

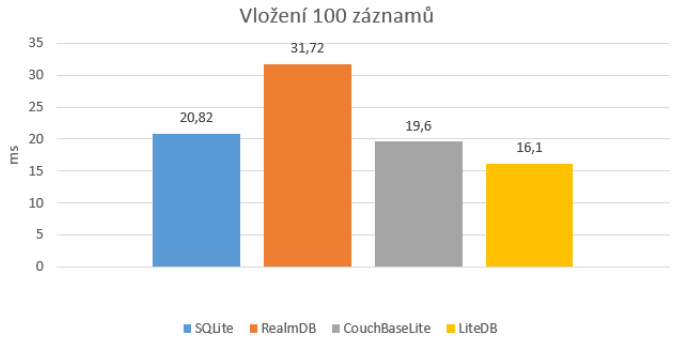
5.1 Vkládání dat

První testovanou části je vkládání záznamů do databáze. V tabulce 4 jsou vidět získané hodnoty z testování v milisekundách.

Počet vkládaných záznamů	SQLite	RealmD	CouchBaseLite	LiteDB
100	20,82	31,72	19,6	16,1
1 000	36,95	226,7	107,88	123,19
50 000	1 117,66	12 070,57	8 581,01	6 268,98

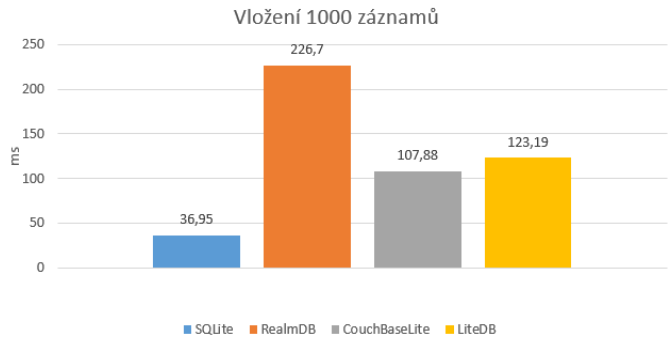
Tabulka 4: Čas vkládání záznamů v milisekundách

Na obrázku 4 je zobrazen graf, kde jsou ve sloupcích uvedeny získané hodnoty pro vkládání sta záznamů do databáze. Z obrázku 10 je tedy patrné, že hodnoty nejsou výrazně odlišné. Vložení sta záznamů je v podstatě okamžité u všech testovaných databázových systémů. Nejlépe si zde vedl systém LiteDB a naopak nejhůře je na tom RealmDB. I když u RealmDB je pro vkládání dat využito BULK INSERT, což znamená, že jsou záznamy vkládány hromadně a navíc je celé vložení prováděno v transakci, je nejpomalejší. CouchBaseLite nevyužívá ani transakci ani hromadné vkládání, a přesto je doba pro vložení sto záznamů průměrná. SQLite i LiteDB pro vkládání využívají také hromadné operace.



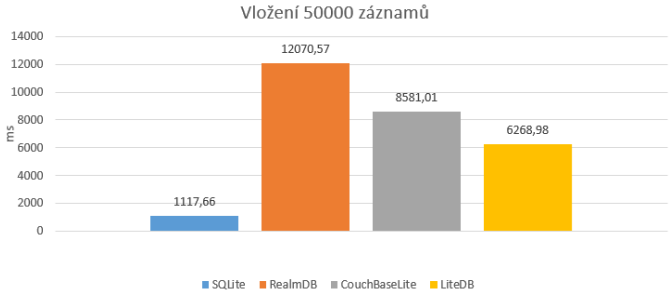
Obrázek 10: Graf pro vkládání 100 záznamů

Na obrázku 11 je zobrazen graf pro vkládání tisíce záznamů do databáze. Zde jsou již rozdíly, jak je vidět značně patrné. Nejlepších výsledků zde dosahuje relační databázový systém SQLite. Na druhém místě je CouchBaseLite, který pro vložení tisíce záznamů potřebuje přibližně 3x tolik času. Nepatrně za CouchBaseLite je RealmDB, který oproti SQLite je 4x pomalejší. Na posledním místě je opět RealmDB, jenž potřebuje na vložení tisíce záznamů 6x tolik času než SQLite.



Obrázek 11: Graf pro vkládání 1000 záznamů

Na obrázku 12 je zobrazen graf, kde jsou zobrazeny hodnoty pro vkládání padesát tisíc záznamů do databáze. I zde si vedl nejlépe relační databázový systém SQLite. Rozdíl mezi nejrychlejším a nejpomalejším systémem je zde skoro jedenácti násobný. I v tomto případě je nejpomalejším systémem RealmDB. Na druhém místě je LiteDB, který je přibližně 6x pomalejší než SQLite a na třetím místě CouchBaseLite, který pro vložení padesáti tisíc záznamů potřebuje skoro osminásobek času.



Obrázek 12: Graf pro vkládání 50000 záznamů

Jak je z grafů patrné, nejlépe si u vkládání dat do databáze vede databázový systém SQLite, který je jednoznačně nejrychlejší, a to i přes fakt, že na rozdíl od ostatních testovaných systémů je počet záznamů 4× větší. Pokud by tedy hlavním kritériem pro aplikaci byla rychlost zápisu dat do databáze, bylo by vhodné použít databázový systém SQLite.

5.2 Selekce dat

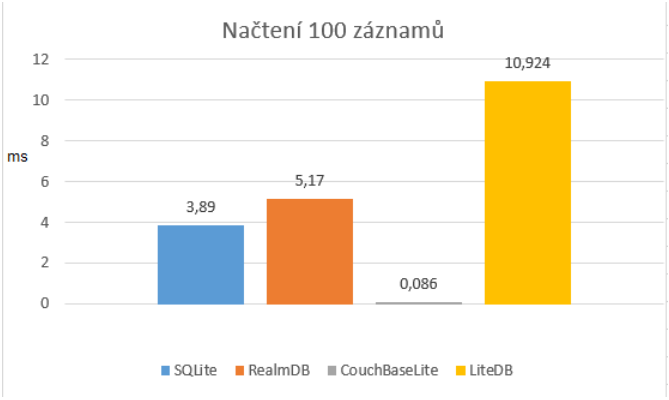
5.2.1 Hromadné načtení dat

V tabulce č. 5 jsou zobrazeny získané hodnoty v milisekundách z testování hromadného načítání záznamů z databáze a následného parsování do kolekce v aplikační vrstvě.

Počet vybíraných záznamů	SQLite	RealmD	CouchBaseLite	LiteDB
100	3,89	5,17	0,086	10,924
1 000	39,75	8,12	0,47	103,19
50 000	1 673,86	123,97	16,31	4 913,11

Tabulka 5: Čas hromadného načtení v milisekundách

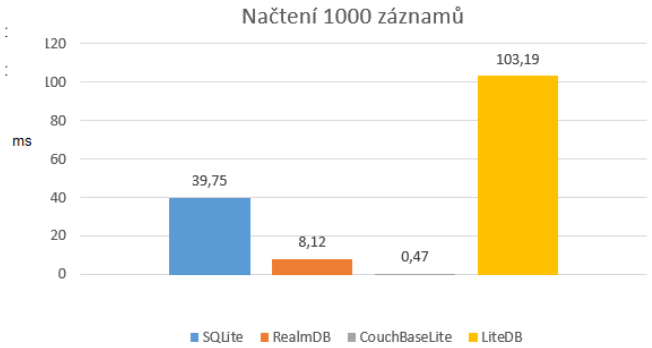
Jak je patrné z obrázku 13, rozdíly pro načtení sta záznamů z databáze jsou značně rozlišné. Zatím co nejrychlejší systém nepotřebuje pro operaci načtení ani milisekundu, nejpomalejší systém jich potřebuje více než 10. V případě sto načítaných záznamů je jasně nejlepší CouchBaseLite. Druhý systém v pořadí je SQLite, který sto záznamů načte za necelé čtyři milisekundy. O něco pomalejší s pěti milisekundami je RealmDB a jednoznačně největší problém s hromadným načítáním má LiteDB.



Obrázek 13: Graf pro hromadné načtení 100 záznamů

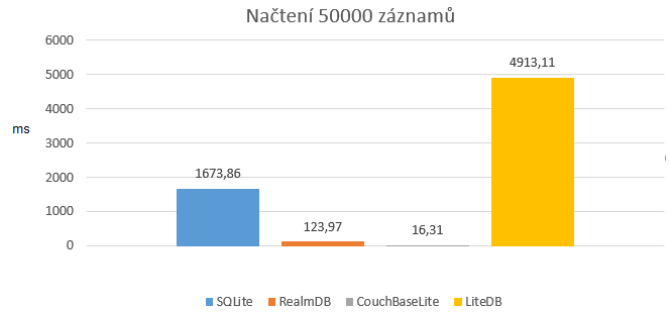
V případě, kdy dochází k načtení jednoho tisíce záznamů, je pořadí systémů na prvním a posledním místě stejné, liší se však v druhém místě, kde je RealmDBnamísto SQLite, které v tomto testu skončilo až na třetím místě. Na obrázku 14 je graf, který tyto data demonstruje. Je tedy vidět, že s rostoucím počtem záznamů je u systémů CouchBaseLite a RealmDB jen nepatrné navýšení času potřebného k načtení dat. V případě RealmDB a SQLite je s rostoucím množstvím dat i přímo úměrně rostoucí čas.

Poslední testovaná kolekce pro hromadné načtení dat z databáze obsahuje padesát tisíc záznamů, jejichž hodnoty získané z testování jsou zobrazeny v grafu na obrázku 15. I v tomto nejlepších výsledků dosáhl databázový systém CouchBaseLite. Pořadí v druhém a třetím testu je



Obrázek 14: Graf pro hromadné načtení 1000 záznamů

stejné a podobně jako tomu bylo v druhém testu, i v tomto je navýšené množství přímo úměrné času potřebnému k načtení dat z databáze.



Obrázek 15: Graf pro hromadné načtení 50000 záznamů

Z testu hromadného načtení dat lze jednoznačně usoudit, že nejrychleji z databáze jsou hromadné data načítány v databázovém systému CouchBaseLite. Rozdíl oproti databázovým systémům SQLite a LiteDB je obrovský. U systému RealmDB jsou doby pro načtení přibližně desetinásobné, ale z důvodu, že se jedná o jednotky milisekund, není rozdíl až takový, jako se může zprvu zdát. V případě aplikace, která by sloužila převážně pro prezentaci velkého množství dat z databáze, je tedy jednoznačné, že vhodnou volbou by byl jeden z databázových systémů CouchBaseLite nebo RealmDB.

5.2.2 Načtení jednoho záznamu

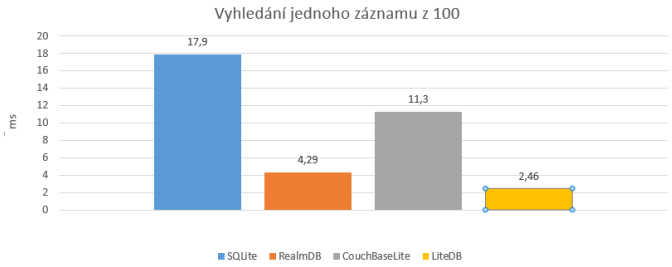
Třetí části testu bylo načtení objednávky dle jejího kódu, který je vždy zároveň primárním klíčem, a který je ve všech případech indexován. V tabulce 6 jsou zobrazeny získané hodnoty v milisekundách z doby potřebné pro načtení jedné objednávky z různých objemů dat a to sto, tisíc a padesát tisíc objednávek.

V prvním případě se jednalo o načtení objednávky s identifikačním číslem 67. Na obrázku 16 jsou zobrazeny časy v milisekundách potřebné pro načtení dané objednávky z celkového

Počet záznamů v databázi	SQLite	RealmDB	CouchBaseLite	LiteDB
100	17,9	4,29	11,3	2,46
1 000	19,26	4,64	23,99	22,76
50 000	39,42	68,73	2 170,68	883,92

Tabulka 6: Čas načtení jednoho záznamu z kolekce v milisekundách

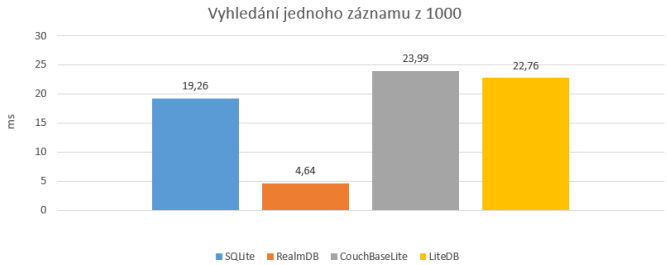
počtu sto záznamů uložených v databázi. Z grafu je patrné, že načtení jednoho záznamu je nejrychlejší v objektových databázích. SQLite v případě tohoto testu je nejpomalejší, a to až 7× oproti nejrychlejšímu systému, který v tomto případě je LiteDB. Po systému LiteDB si nejlépe s načtením jedné objednávky vedl systém RealmDB, který pro načtení jedné objednávky potřebuje dvojnásobek času. Na třetím místě skončil CouchBaseLite, u kterého je čas pro načtení jedné objednávky přibližně 4× větší. Ve všech případech se jedná o jednotky milisekund, ale jedná se o malé objemy dat, z kterých je objednávka vybírána.



Obrázek 16: Graf pro načtení jednoho záznamu ze 100

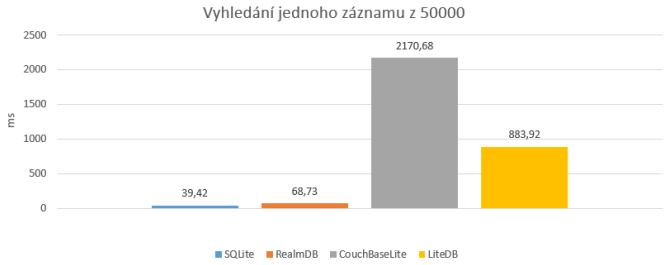
V druhém testu se vybírá jedna objednávka s identifikačním číslem 472 z celkového počtu jednoho tisíce objednávek. Na obrázku 17 je zobrazen graf, na kterém jsou zobrazeny časy v milisekundách pro načtení jedné objednávky z tisíce. I v tomto případě nejrychleji byla objednávka načtena systémem RealmDB, a to se stejným časem jako v případě vyhledávání ve sto záznamech. Na druhém místě se těsně před třetím systémem LiteDB umístil SQLite a na posledním místě je CouchBaseLite. Rozdíly mezi druhým a posledním jsou ale minimální a dalo by se tedy říct, že jsou na tom tyto systémy stejně. S přibývajícím počtem záznamů je patrné, že největší nárůst času byl zjištěn u systému LiteDB, který pro 10× větší množství záznamů potřebuje 10× tolik času.

Poslední test v rámci načtení jedné objednávky bylo načtení objednávky s identifikačním číslem 4762 z celkového počtu padesáti tisíc objednávek. V tomto případě nejlepšího času dosahuje systém SQLite, u které oproti předchozímu testu došlo k navýšení času jen o dvojnásobek, i když množství záznamu je 50× větší. Na druhém místě je s dvojnásobným časem oproti systému SQLite systém RealmDB u kterého oproti druhému testu se čas pro načtení objednávky navýšil 15×. Na třetím místě skončil s více jak dvacetinásobným časem oproti první systém LiteDB. V tomto případě došlo oproti druhému testu k navýšení času o čtyřiceti násobek. A poslední



Obrázek 17: Graf pro načtení jednoho záznamu z 1000

skončil systém CouchBaseLite, který je 55× pomalejší než SQLite. V tomto případě se oproti druhému testu čas navýšil 90×.



Obrázek 18: Graf pro načtení jednoho záznamu z 50000

V rámci tohoto testu jsou výsledky pro různé kolekce opravdu rozlišné. V případě malých aplikací, ve kterých by bylo hlavní funkcí pracovat s jednotkami záznamů, by bylo vhodné použít RealmDB . U větších aplikací, kdy objemy dat se budou pohybovat v tisících a více záznamech je volbou SQLite, z důvodu, že s rostoucím množstvím záznamů v databázi jen nepatrně roste čas pro vyhledání vybraného záznamu.

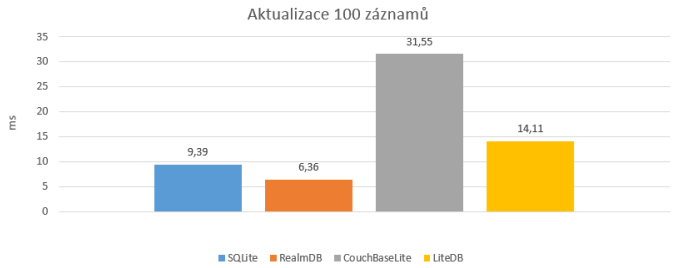
5.3 Aktualizace dat

Předposlední testovanou části je hromadná aktualizace dat. Opět probíhá ve třech testech podle velikosti testované kolekce, a to opět ve stejných počtech sto, tisíc a padesát tisíc záznamů. V tomto případě je na každé aktualizované objednávce změněna ceny objednávky. V tabulce 7 jsou časy v milisekundách potřebné k aktualizaci daného počtu záznamů. Z pohledu na čísla je patrné, že rozdíly mezi jednotlivými kolekcemi dat jsou obrovské.

Počet aktualizovaných záznamů	SQLite	RealmDB	CouchBaseLite	LiteDB
100	9,39	6,36	31,55	14,11
1 000	40,52	6,93	275,29	127,3
50 000	1 602,65	62,84	18 758,59	6 390,01

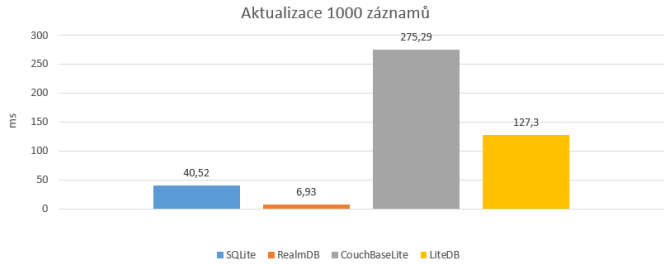
Tabulka 7: Čas potřebný pro aktualizace v milisekudnách

Na obrázku 19 je graf, který zobrazuje počet v milisekundách potřebných k aktualizaci daného počtu záznamu v databázi. V tomto případě, kdy je aktualizované sto objednávek je vítězem RealmDB. Na druhém místě s o něco horším časem je SQLite, následovaný systémem LiteDB. Posledním systémem v tomto případě je CouchBaseLite s časem přes třicet jedna milisekund. Rozdíly mezi SQLite, RealmDB a LiteDB nejsou tak rozlišné.



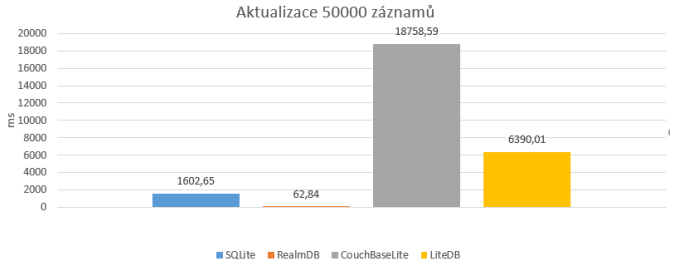
Obrázek 19: Graf pro aktualizaci 100 záznamů

V druhém testu se testovala aktualizace jednoho tisíce záznamů. Výsledky tohoto testu jsou zobrazeny na obrázku 20, kde je zobrazeno pořadí systémů, které se oproti prvnímu testu nezměnilo. U systému RealmDB došlo pouze k minimálnímu nárůstu času potřebného pro aktualizaci objednávek. U systému SQLite se čas navýšil přibližně 4×. U systému LiteDB a CouchBaseLite se čas navýšil přímo úměrně zvýšenému množství dat. S touto tendencí jsou systémy LiteDB a CouchBaseLite oproti RealmDB a SQLite značně pomalé, ale o tom více prozradí poslední část testu.



Obrázek 20: Graf pro aktualizaci 1000 záznamů

V třetím testu se testovala aktualizace padesáti tisíců záznamů. I v tomto testu se pořadí oproti prvnímu nezměnilo, jak je vidět z obrázku 21. Zde jsou již zobrazeny vysoké hodnoty, kdy aktualizace padesáti tisíc záznamu trvá v nejhorším případě skoro devatenáct vteřin a to u systému CouchBaseLite, který je oproti předchozímu testu skoro 70× pomalejší, takže tendence přímé úměry s rostoucím množstvím dat byla překročena. Třetí v pořadí skončil systém LiteDB, který byl o proti druhému testu 50× pomalejší a udržel si tendenci přímé úměry s rostoucím množstvím dat. Druhý v pořadí skončil SQLite, u kterého došlo k nárůstu o proti druhému testu o čtyřiceti násobek, a tak i zde dochází k tendenci přímé úměry s rostoucím počtem dat. Nejrychleji aktualizuje data systém RealmDB, který pro aktualizaci padesáti tisíc záznamů potřeboval pouze šedesát tři milisekund, což je oproti CouchBaseLite skoro 300× rychleji.



Obrázek 21: Graf pro aktualizaci 50000 záznamů

Jak je z tohoto testu patrné pro aktualizaci malého počtu dat nehraje roli výběr databázového systému, avšak při větších objemech dat je jasnou volbou databázový systém RealmDB, který v tomto testu jednoznačně dosahuje nejlepších výsledků. Pokud by bylo zapotřebí vytvořit aplikaci, ve které by hlavní úlohou bylo aktualizovat velké množství záznamů v malých časových intervalech určitě by bylo vhodné pro datovou vrstvu využití systému RealmDB.

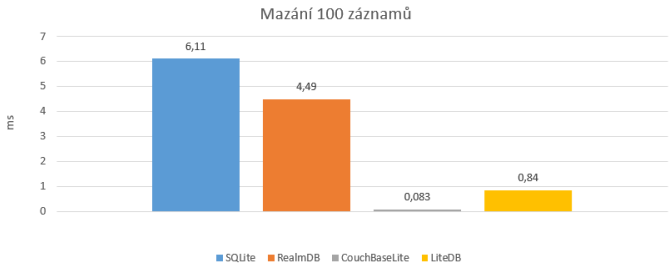
5.4 Mazání dat

Poslední části testu je mazání záznamů. V tomto případě proběhne testování v souvislosti mazání celé kolekce dat, se kterou byly prováděny předchozí testy. V případě relační databáze to znamená, že musí být smazány veškerá data z tabulek objednávka, položka objednávky, zákazník a sortiment. V případě dokumentových databází je samotný princip jednoduší, a to tím, že dojde ke smazání celého dokumentu. V tabulce 8 jsou naměřené časy potřebné pro smazání různých kolekcí dat. Z dat v tabulce je jasně patrné, že mazání dat v relační databázi bude vždy pomalejší v případě, že dochází ke smazání celé kolekce dat.

Počet mazaných záznamů	SQLite	RealmDB	CouchBaseLite	LiteDB
100	6,11	4,49	0,083	0,84
1 000	27,16	6,75	0,5	2,11
50 000	1071,87	9,6	11,71	2,61

Tabulka 8: Čas potřebný pro mazání dat v milisekudnách

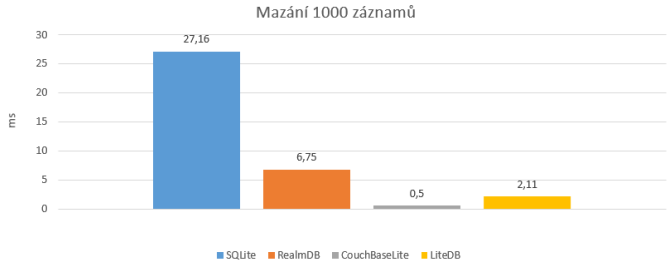
V první testu bylo smazáno sto záznamů a časy potřebné pro tuto operaci jsou zobrazeny na obrázku 22. SQLite je ve všech těchto testech nejhorší, a to z důvodu, že musí promazávat čtyři tabulky. V prvním případě se jedná o sto záznamů a čas pro mazání mezi prvním systémem (CouchBaseLite) a posledním (SQLite) je více jak sedmdesáti násobný. U systému CouchBaseLite je smazání sta záznamů v podstatě okamžité, stejně jako u LiteDB. V případě RealmDB je třeba pro smazání sta záznamů čtyři milisekundu a u SQLite je to milisekund 6. Ani v jednom případě tedy čas potřebný pro smazání malého počtu záznamů nepřesahuje hranici deseti milisekund. Pro volbu aplikace, kdy k mazání dat dochází jen zřídka, není tato vlastnost důležitá.



Obrázek 22: Graf pro mazání 100 záznamů

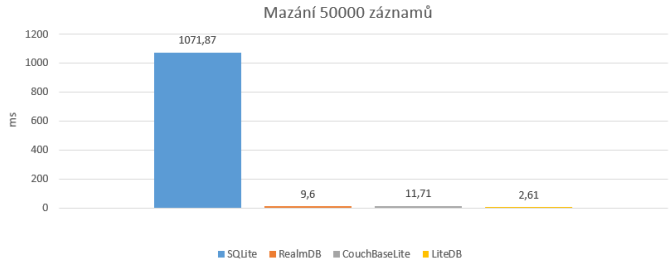
V druhém testu se testovalo mazání tisíce záznamů a jak je z obrázku 23 patrné, SQLite je opět nejpomalejší a pro smazání desetinásobného množství dat oproti prvnímu testu potřebuje více jak 4× více času. Třetí je v tomto případě RealmDB, u kterého čas potřebný pro smazání nepatrně narostl oproti prvnímu testu. Druhým v pořadí skončil systém LiteDB, u kterého také došlo k nepatrnému navýšení času potřebného pro smazání tisíce záznamů oproti sto záznamům.

A nejrychlejším je opět systém CouchBaseLite, u kterého se čas potřebný pro smazání tisíce záznamů opět nedostal nad čas jedné milisekundy.



Obrázek 23: Graf pro mazání 1000 záznamů

V posledním testu se testovalo mazání padesáti tisíci záznamů. Na obrázku 24 opět je patrné, že nejhůře dopadl SQLite, a to s časem přes jednu sekundu. Na třetím místě skončil dosud vedoucí CouchBaseLite, u kterého se po navážení množství dat navýšil čas více jak dvaceti násobně. Druhým v pořadí skončil systém RealmDB s časem okolo 6 milisekund. Nejrychleji si se smazáním padesáti tisíc záznamů poradil systém LiteDB, který pro tuto operaci potřebuje skoro stejný čas jako pro smazání jednoho tisíce záznamů.



Obrázek 24: Graf pro mazání 50000 záznamů

Jelikož operace mazání se nevyužívá tak často jako jsou operace selekce, aktualizace či vkládání, a časy potřebné pro tuto operaci jsou ve všech případech kromě SQLite velmi podobné, tak by v případě aplikace, u které se vyžaduje časté mazání velkého množství záznamů, bylo vhodné využít jeden z dokumentových databázových systémů.

5.5 Zhodnocení testování

Z celkového pohledu testování probíhalo u všech systémů bez problému, a nebylo třeba žádné zvláštní implementace ani nástrojů, aby bylo zajištěno stejných podmínek pro všechny testované systémy. V případě implementace testování by bylo vhodnější testovací aplikaci zautomatizovat, a výsledky měření zapisovat do textové podoby, čímž by odpadla nutnost provádět testování. V návrhu aplikace bohužel tento fakt byl opomenut, a tak testování probíhalo manuálně pro každý systém zvlášť, ale vždy nad stejnou kolekcí dat. Z výsledků je patrné že ani jeden z testovaných systémů nedosahuje ve všech částech nejlepších časů. Z pohledu návrhu databáze a

aplikace jsou zjištěné data přínosné, neboť každá aplikace je jedinečná a má různé nároky na práci s daty. Jednoznačně vyhlásit nejlepší systém nelze a to z mnoha důvodu. Jedním z nich je, že různé časové prodlevy u některých operací, jako je vkládání dat do databáze, lze řešit jiným vláknem, kdy nemusí aplikace čekat na provedení operace. Z tohoto důvodu je u každého systému zhodnocení silných a slabých stránek. V případě systémů, které budou pracovat s daty v řádu stovek záznamů, nejsou rozdíly tak velké, jako je tomu u množin dat v řádech tisíců a více a proto ve vyhodnocení jednotlivých systémů je přihlíženo k velkým objemům dat.

5.5.1 SQLite

Databázový systém SQLite si v testu vedl dobře především v oblasti vkládání dat do databáze. V tomto ohledu relační databázový systém byl o hodně rychlejší než jeho dokumentoví soupeři. Čas, který je potřebný pro operace vkládání je mnohonásobně kratší, než u ostatních systémů. V případě načítání dat z databáze je v případě databázového systému SQLite velký rozdíl mezi načítáním jednoho záznamu a hromadného načítání dat z databáze. Zatím co u načtení jednoho záznamu SQLite dosahoval nejlepších výsledků, tak u hromadného načítání si vedl po LiteDB nejhůře. U aktualizací dat si SQLite nevedl nejhůře, ale oproti RealmDB zaostává, naopak oproti systémům CouchBaseLite a LiteDB si vedl velmi dobře. V poslední částí testu je tento systém nejhorším a určitě není vhodné jej použít tam, kde je největší důraz kladen na mazání dat. Z celkového pohledu tedy lze říct, že databázový systém SQLite je vhodné použít v aplikacích, od kterých se očekává časté vkládání dat o velkém počtu, a zároveň časté dotazování jedinečných záznamů. Z pohledu hromadného načítání je zde možnost načíst data jednou a předat je aplikační vrstvě. Určitě se však nehodí pro časté načítání velkého objemu dat. V případě častých aktualizací dat není nejlepší volbou, ale pokud by se jednalo jen o sekundární funkce, není SQLite určitě tou nejhorší volbou.

5.5.2 RealmDB

RealmDB si nejlépe vedl v oblasti aktualizace dat, ale ani v ostatních ohledech na tom není vůbec špatně, až na vkládání, u kterého je jednoznačně nejpomalejší. Z pohledu vkládání dat do databáze, lze říct, že není vhodné tento systém použít tam, kde se často vkládají velké objemy dat, a zároveň je potřebné okamžitě pracovat dále s daty v databázi. V oblasti načítání dat z databáze si v obou testovaných částech vedl dobře a skončil v obou případech na druhém místě. Nejedná se ale o žádné velké rozdíly mezi nejrychlejšími systémy v rámci testování. V případě aktualizací dat je RealmDB naprostým vítězem, neboť časy, kterých dosahuje, jsou rychlejší v řádech desítek až stovek násobku oproti ostatním testovaným systémům. Samotné mazání dat je u všech dokumentových systémů, které byly testovány, je prakticky stejně rychlé. V celkovém pohledu je tedy RealmDB vhodné použít u aplikací, které data načtou jednou, a poté se s nimi často pracuje. Databázový systém RealmDB určitě není vhodné použít tam kde je nutné časté vkládání dat do databáze.

5.5.3 CouchBaseLite

Vůbec nejhůře si v testování vedl asi systém CouchBaseLite, který vyniká jen v hromadném načítání dat z databáze. Toto však může být velkou výhodou u aplikací, kde jsou data zaznamenávány postupně, ale je potřeba často potřeba číst velké množství dat z databáze. V oblasti načítání jednoho záznamu je lepší použít určitě jiné systémy. Vkládání nových záznamů do databáze není nejsilnější stránkou tohoto systému, ale pokud by aplikace nevkládala tisíce záznamu v častých intervalech určitě by tento systém byl vhodný. Pro potřeby aktualizovat data je CouchBaseLite tou nejhorší volbou. Časy, které jsou třeba pro aktualizaci, jsou zbytečně velké, a tak pro aplikace s častou potřebou aktualizovat data se tento systém nehodí. Mazání dat je pro tento systém celkem bezproblémové.

5.5.4 LiteDB

Systém LiteDB není žádnou výhrou ve volbě databázového systému, kromě mazání nevyniká v žádném testu při větším objemu dat. V problematice mazání je sice nejrychlejší, ale naměřené časy jsou si na časové ose tak blízko, že hodnotit to nemá ani smysl. Jediným překvapením je, že mezi dokumentovými databáziemi je nejrychlejší pro operace vkládání dat do databáze v ostatních oblastech zatím, ale zaostává. V případě aplikací, u kterých dochází k častému vkládání dat do databáze a podmínky využití dokumentové databáze, je LiteDB správným řešením.

6 Závěr

Cílem této práce bylo nastudovat problematiku embedded databázových systému určených pro mobilní zařízení, a to jak relačních, tak dokumentových využívající modelu klíč-hodnota, a provést jejich klasifikaci a porovnání oproti databázovým systémům s architekturou klient-server a následně je otestovat. V úvodu práce je vysvětleno několik základních pojmů, které je nezbytné pochopit pro porozumění problematiky této práce. V rámci další části se čtenář dozví základní kritéria pro výběr testovaných databázových systémů, které jsou mu následně představeny. Ještě před samotným vyhodnocením testování je popsán samotný princip, kterým jsou databázové systémy testovány a technická specifikace aplikace a technologie, které pro její vývoj byly použity a zároveň také zařízení, na kterém testy probíhaly. Samotné vyhodnocení testování je rozděleno do pěti částí, a to dle operací, které jsou v rámci testu prováděny. Pro každou část testu je vždy zobrazena tabulka s naměřenými hodnotami potřebnými pro vykonání daných operací a následně také grafické porovnání mezi jednotlivými testovanými systémy. V závěru je ke každému z vybraného systému věnováno vyhodnocení a klasifikace, pro jaký typ aplikací je tento embedded databázový systém pro mobilní zařízení vhodný. V rámci testování byly vybrány systémy CouchBaseLite, LiteDB, RealmDB a SQLite, které byly otestovány pomocí vytvořené testovací aplikace, a u kterých bylo zjištěno, že pro aplikace, které potřebují často vkládat nová data je vhodné využit databázový systém SQLite, naopak pro aplikace, kde je zapotřebí často data aktualizovat je nejvhodnější využít databázový systém RealmDB. Pro častou selekci velkého množství dat se nejvíce hodí databázový systém CouchBaseLite. Výběr jednoho záznamu je nejrychlejší u databázového systému SQLite a pro operace mazání docházelo během testování k podobným výsledům.

Hlavním přínosem této práce je klasifikace embedded databázových systémů určených pro mobilní zařízení z pohledu času potřebného pro provedení základních operací nad daty. Po přečtení této práce je čtenář schopen dle výsledků a doporučení v podobě zhodnocení výsledku jednotlivých systémů vybrat vhodný embedded databázový systém pro aplikaci dle požadavků a předpokladů, které v rámci návrhu aplikace byly stanoveny.

Literatura

[1] VOSTROVSKÝ, Václav. Relational database systems. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2011. ISBN 978-80-213-2215-8.

[2] Relační databáze: Distanční výuková podpora [online]. Dvořákova 7, 701 03 Ostrava, 2006 [cit. 2018-04-22]. Dostupné z: http://fakulty.osu.cz/prf/rsk/uploaded/1942_XRELA1.pdf

[3] SULLIVAN, Dan. NoSQL for Mere Mortals. 1. vyd. Upper Saddle River: AddisonWessley,2015. Kindle edition. ISBN 978-0-13-402231-2.

[4] Embedded databáze - Úvod. Www.root.cz [online]. 2004 [cit. 2018-04-22]. Dostupné z: https://www.root.cz/clanky/embedded-databaze-uvod/

[5] SQLite [online]. 2018 [cit. 2018-04-22]. Dostupné z: https://www.sqlite.org/

[6] Realm [online]. 2018 [cit. 2018-04-22]. Dostupné z: https://realm.io/

[7] Couchbase [online]. 2018 [cit. 2018-04-22]. Dostupné z:https://www.couchbase.com/

[8] LiteDB [online]. 2018 [cit. 2018-04-22]. Dostupné z: http://www.litedb.org/

[9] KRÁTKÝ, Michal a Radim BAČA. Databázové systémy [online]. [cit. 2018-04-22]. Dostupné z : http://dbedu.cs.vsb.cz/SubPages/OpenFile.aspx?file=book/dbcb.pdf

[10] ŠIMŮNEK, Milan. SQL: kompletní kapesní průvodce [online]. Vyd. 1. Praha: Grada, 1999 [cit. 2018-04-22]. ISBN 80-716-9692-7.

[11] Alex Berson, CLIENT/SERVER ARCHITECTURE. 2nd edition, McGraw-Hill, 1996. ISBN 9780070050761

[12] K.V.K Prasad, Embedded / Real-Time Systems: Concepts, Design and Programming Black Book, ISBN ISBN: 9788177224610

[13] A Relational Model of Data for Large Shared Data Banks [online]. 1970 [cit. 2018-04-29]. Dostupné z: http://db.dobo.sk/wp-content/uploads/2015/11/Codd_1970_A_relational_model.pdf

[14] Databázové systémy 2: Studijní opora. Jihlava. Jihlava: Vysoká škola polytechnická Jihlava, Tolstého 16, Jihlava, 2014. ISBN 978-80-87035-89-4.

[15] SCALZO, Bert, Kevin KLINE, Claudia FERNANDEZ, Donald K. BURLESON a Mike AULT. Database Benchmarking Practical methods for Oracle and SQL Server. ISBN 978-0977671533.